CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Desarrollo de una herramienta de monitorización y control de aplicaciones distribuidas en un entorno de integración continua

Autor: Carlos David Rodríguez Peña

Director: Javier Soriano Camino

MADRID, JUNIO DE 2014

Agradecimientos

Me gustaría dedicar unas pocas líneas a todas esas personas que han estado conmigo apoyándome y ayudándome en la medida de lo posible a completar este trabajo, y no sólo eso, sino a completar todo el duro camino que me ha llevado a al fin poder estar en este punto de mi carrera.

A todos los profesores de la Facultad de Informática de la UPM que me han enseñado la mayor parte de lo que sé del campo de la informática, y en especial a mis tutores y jefes de laboratorio, Javier Soriano y Genoveva López, por confiar en mí y darme la gran oportunidad de trabajar con ellos estos últimos años.

A mis amigos y compañeros de la Facultad, especialmente a mis compañeros de laboratorio Salva, Alberto, David y Fernando, por ayudarme a aprender muchísimo sobre diferentes campos y por esos ratos tan buenos que hemos pasado juntos.

A mis padres, por haber estado siempre conmigo, proveyendo todos los medios necesarios para mi educación, asegurándose de que nunca me faltase de nada y ayudándome a conseguir mis metas. A mi hermano por soportar mis bromas y por los buenos ratos que pasamos juntos. A mis abuelos por preocuparse siempre de mí, y asegurarse de que iba en el buen camino.

Y sobre todo quiero darle las gracias Sonia, mi novia, por estar siempre ahí para cualquier cosa, ayudándome y apoyándome en los momentos más duros, cuando creía que toda esperanza estaba perdida. Gracias por no dejar nunca de creer en mí.

Muchas gracias a todos, de corazón.

Sin vosotros nunca habría llegado a donde estoy ahora.

Resumen

Debido a la creciente relevancia de la computación en la nube y de los sistemas distribuidos, cobran también creciente interés las herramientas que ayudan a los desarrolladores y administradores a desempeñar sus funciones con la mayor eficacia posible.

Por ello el objetivo principal de este trabajo es el desarrollo de una herramienta capaz de crear y controlar un entorno de almacenamiento de claves distribuidas desde una máquina local e independiente, aumentando la productividad mediante la automatización de todas las tareas. La herramienta desarrollada tiene la capacidad necesaria para integrarse tanto en proyectos que se encuentren en marcha como para proyectos que aún no hayan comenzado y proveer una solución sencilla, eficaz, y, sobre todo, útil.

A lo largo del trabajo se ha realizado una gran tarea de análisis para determinar cuáles serán, de entre las posibilidades existentes, las más apropiadas para su implementación, teniendo en cuenta las tecnologías líderes disponibles en el estado del arte. Ello ha requerido también la obtención de una mejor comprensión de su funcionamiento interno. Se han realizado diferentes diseños que se han analizado y discutido en detalle para encontrar la solución que mejor se adaptaba a los objetivos propuestos. Y finalmente se ha desarrollado una herramienta ligera y sencilla, pero con un gran potencial para la administración.

Abstract

Due to the growing relevance of cloud computing and distributed systems it seems interesting to take into account the importance of the administration tools that help developers and administrators fulfill their duties in the most efficient ways.

Because of this motivation, the main objective of this project is the development of a tool capable of creating and controlling a distributed key storing environment from a local and independent machine, improving the productivity thanks to the automation of all the involved tasks. The developed tool is able to integrate itself into already running projects as well as in not-yet-started ones, providing a simple, efficient and overall useful solution.

During this project big tasks of research and analysis have taken place in order to determine, from the existent possibilities, the most suitable for its implementation, taking into account the leading technologies in the sector, which are described in the state of the art section. This has required the acquisition of a better insight of their inner workings. Some different designs have been made and have been discussed in detail with the intention of finding the solution that best suits the proposed objectives. And finally a lightweight and simple tool has been developed, which presents a very big potential for administration tasks.

Índice

Índice de figuras

1. Introducción

En los últimos años las aplicaciones distribuidas han experimentado un gran crecimiento gracias al apogeo de los sistemas en la Nube (Cloud Computing). El mayor uso de los dispositivos móviles, así como el gran número de plataformas que se ofrecen como servicio no ha hecho sino impulsar el uso de estas tecnologías a unos niveles que no se habrían previsto hace una década. Como se puede observar en la gráfica de la *Figura 1*, el uso de Internet ha experimentado un crecimiento muy importante en los últimos años, llegando a unas cifras de usuarios realmente elevadas.

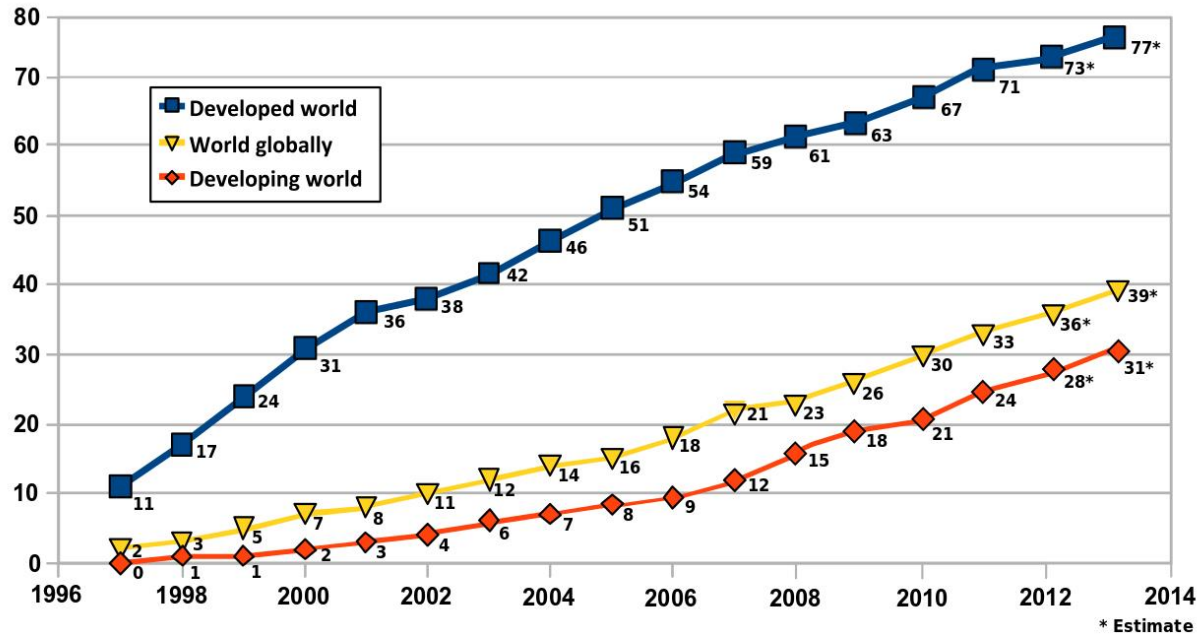

Figura 1. Crecimiento del porcentaje de usuarios de Internet. Fuente: Wikipedia

A día de hoy, las grandes empresas de Tecnologías de la Información se están centrando cada vez más en la provisión de servicios distribuidos entre grandes redes de máquinas

situadas en diferentes partes del territorio, en lugar de estar concentradas en un centro de datos físico. Este hecho ha llevado al desarrollo de algunas herramientas para ayudar a la realización de los despliegues en estas redes. Sin embargo, al no ser esta la prioridad de las empresas, estas herramientas en muchas situaciones resultan incómodas y poco eficientes desde el punto de vista del desarrollador. Además, muchas de ellas se desarrollan como proyectos aislados y ven su desarrollo interrumpido cuando surgen temas más urgentes, o limitado a una serie de sistemas concretos.

La idea de este Trabajo de Fin de Grado surge en el marco del Laboratorio de Innovación Abierta UPM – Telefónica Digital con el propósito de dar una solución a la problemática descrita sobre estas líneas, y su objetivo es el desarrollo de unas herramientas que sean capaces de aprovechar las capacidades de las diferentes soluciones existentes y automatizar sus funciones de tal forma que se libere en la medida de lo posible el trabajo necesario por parte del desarrollador, para que pueda emplear su tiempo en lo que es realmente importante. El trabajo en concreto se ha centrado en la gestión de los elementos de configuración de las máquinas físicas que participan en las aplicaciones distribuidas de forma centralizada, asegurando que todas ellas utilizan los mismos parámetros y que las modificaciones son transparentes de cara al desarrollador, de tal modo que las máquinas funcionen y sean capaces de reaccionar adecuadamente y de manera instantánea ante los cambios que se lleven a cabo. Además, esta herramienta va a estar complementada por otra herramienta que se está desarrollando en el mismo laboratorio y que se encargará del despliegue de nuevas máquinas y de la gestión del estado de las mismas.

1.1 Objetivos

El objetivo principal de este trabajo, como se ha mencionado en el apartado anterior, es la implementación de una herramienta de productividad que integre soluciones existentes y permita su utilización centralizada y de manera transparente por un desarrollador.

Los objetivos específicos del trabajo vienen descritos a continuación:

- **Investigación comparativa de las soluciones existentes**: En primer lugar es necesario conocer lo que ofrecen las principales soluciones existentes, y juzgar

cual es la que mejor se adapta a las necesidades del proyecto en términos tanto de eficiencia, como de facilidad de uso. Para ello se tomarán en cuenta las tecnologías líderes en el mercado en la materia que nos atañe, y que se han mantenido en ese puesto durante varios años a día de hoy.

- **Aprendizaje y experimentación con la solución escogida**: De este modo se realizarán una serie de pruebas en simulaciones de casos reales, para conocer sus capacidades más a fondo, así como sus limitaciones y su modo de operar. También se le dará una gran importancia a la facilidad de uso de la herramienta, haciendo hincapié en las necesidades de mantenimiento y despliegue.
- **Diseño de una solución apropiada**: Es necesario tener en cuenta que el objetivo de la herramienta que se desea diseñar es simplificar el trabajo del desarrollador a la hora de realizar un despliegue, de tal forma que las modificaciones en la configuración tanto en la fase de desarrollo y pruebas, como durante el paso a producción y las actualizaciones posteriores, se hagan de la forma más transparente posible, evitando la molestia de tener que gestionar cada máquina individualmente.
- **Implementación de la herramienta**: Se desarrollarán todas las partes de la herramienta al mismo tiempo y de manera incremental, para poder tener una funcionalidad básica y probable cuanto antes, y ser capaz de hacer evolucionar la herramienta a partir de dicha base. El principal objetivo es conseguir dicha versión básica que demuestre las capacidades de la herramienta y desarrollarla de tal forma que sea posible y sencillo ampliar la herramienta en un futuro, ya sea añadiendo nueva funcionalidad, mejorando su seguridad o incluyendo nuevas posibilidades de mejora, como técnicas de alta disponibilidad o de tenencia múltiple.
- **Pruebas de la herramienta en un entorno simulado**: Finalmente, una vez completado el desarrollo de la herramienta, se realizarán una serie de pruebas en un entorno que simule una situación real, lo que permitirá demostrar las capacidades de la herramienta desarrollada. Estas pruebas se limitarán a demostrar las capacidades de la herramienta en un entorno sin ningún tipo de condiciones de estrés externo, pero servirán para demostrar el concepto y el funcionamiento de la herramienta en un entorno real.

Es de destacar que se han conseguido todos estos objetivos y que se dispone ya de una herramienta de productividad que ha sido entregada a Telefónica I+D para que pueda analizarse su uso en un entorno real y pueda determinarse su *roadmap* de evolución en base al *feedback* recibido.

1.2 Organización del resto del documento

En este apartado se describe brevemente el contenido del resto de capítulos de esta memoria.

En el Capítulo 2 se tratan todas las tecnologías utilizadas para la realización del proyecto, incluyendo tanto el lenguaje de programación como los sistemas externos utilizados.

En el Capítulo 3 se realiza un estudio del estado del arte acerca de los sistemas distribuidos y los algoritmos de consenso para el almacenamiento de datos en entornos distribuidos, por su relevancia en el diseño de la solución propuesta. También se explicarán las tecnologías evaluadas para la realización de este proyecto.

En el Capítulo 4 se explica en detalle el proceso de creación de la herramienta que ha sido desarrollada en este trabajo, comenzando por una breve explicación sobre el origen de la idea, seguida por el diseño de la herramienta para finalmente mostrar una especificación detallada del producto final.

En el Capítulo 5 se realiza un análisis de los resultados y se exponen las conclusiones obtenidas durante el desarrollo de la herramienta. Finalmente se evalúan las posibles líneas futuras a seguir, proponiendo nuevas funcionalidades y otros detalles que parezcan de interés para la herramienta y que se han quedado fuera de los objetivos propuestos en este trabajo.

2. Tecnologías utilizadas en el desarrollo

En este capítulo se ofrece una rápida revisión de las tecnologías utilizadas para la implementación de esta herramienta. Estas tecnologías son: Python, Fabric y AWS.

2.1 Python

Python fue creado en el año 1991 por Guido van Rossum y representa un lenguaje de programación de propósito general de alto nivel. Es un lenguaje interpretado, esto quiere decir que en lugar de existir un compilador que convierta el código original a código de más bajo nivel, o código objeto, antes de ser ejecutado, como se muestra en la *Figura 2,* se utiliza un intérprete. Un intérprete va leyendo el programa y procesándolo poco a poco, al mismo tiempo que lo ejecuta, como se puede observar en la *Figura 3.* Esto permite que Python se ejecute en modo interactivo desde una consola.

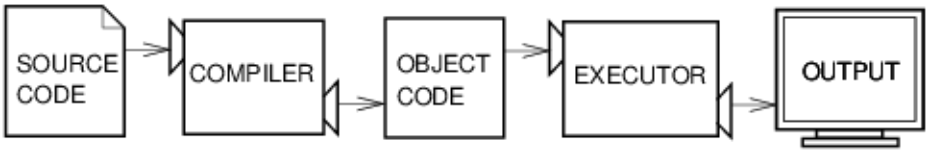

Figura 2. Ejemplo de ejecución de lenguaje compilado. Fuente: Think Python, 2012

Figura 3. Ejemplo de ejecución de lenguaje interpretado. Fuente: Think Python, 2012

Python soporta múltiples paradigmas de programación, como son la programación orientada a objetos, la imperativa, la funcional o la programación por procedimientos. Utiliza tipado dinámico y un recolector de basura para gestionar la memoria. Para

desarrollar en este lenguaje se recomienda seguir las indicaciones de la *Python Enhancement Proposal* (PEP), consejo que se ha seguido al pie de la letra en este proyecto [1].

Este lenguaje, a pesar de ser tan antiguo, se utiliza actualmente en multitud de proyectos por su gran flexibilidad y estabilidad. Además dispone de una enorme y altamente activa comunidad de desarrolladores, lo que hace que desarrollar en este lenguaje sea muy sencillo y amigable. Además Python dispone de muchas implementaciones, lo que permite ejecutarlo en gran variedad de entornos sin ningún problema de compatibilidad. Estas características entre otras han llevado a que se haya incluido en los planes tecnológicos de multitud de empresas, como es el caso de Telefónica I+D, para la que este proyecto va dirigido.

2.2 Fabric

Fabric es una librería de Python que sirve para automatizar la administración y despliegue de máquinas remotas mediante SSH desde una máquina local. Proporciona un conjunto básico de herramientas para ejecutar mandatos de la *shell* de forma tanto local como remota, permitiendo su ejecución vía sudo e incluso permitiendo también tanto la subida como la descarga de ficheros, así como la capacidad para solicitar nuevos datos al usuario, por ejemplo opciones de programas interactivos.

Esta herramienta se utilizará para realizar toda la comunicación con las máquinas remotas que se gestionarán en el proyecto, ya sea para instalar nuevas dependencias o software, o para darles instrucciones como si se tratara de máquinas locales.

Para utilizar la herramienta lo único necesario es crear un archivo, denominado *fabfile*, que contiene una serie de funciones Python. Cada una de estas funciones representa y da nombre a las acciones que podremos realizar en las máquinas remotas y reciben el nombre de "*tasks*" (o tareas en inglés). Aquí podremos definir cada uno de los pasos que queremos que se realicen en la *shell* remota, o bien enviar un script completo y ejecutarlo allí, o incluso solicitar una conexión ssh que abra una consola remota directamente en la máquina y comunicarnos como si se tratara de una llamada ssh normal y corriente.

A continuación se muestra un pequeño ejemplo con una llamada sencilla en Fabric:

```
from fabric.api import run
def host_type():
    run('uname -s')
```

Una vez definidas las tareas deberemos llamarlas una por una según cual sea la que deseamos ejecutar. Para esto existen dos opciones principalmente:

- **Utilizar la herramienta de línea de comandos *fab*:** *fab* es una herramienta que viene incorporada con la instalación de Fabric y que nos permite llamar a las diferentes acciones definidas por su nombre. Esto sería tan sencillo como:

  ```
  $ fab -H localhost host_type
  [localhost] run: uname -s
  [localhost] out: Linux

  Done.
  Disconnecting from localhost... done.
  ```

- **Importar a un código Python:** se podrán importar tanto los elementos de Fabric como las tareas definidas en el *fabfile* y utilizarlas de la misma manera que si fueran funciones normales de Python.

Una de las principales ventajas de Fabric es su capacidad para trabajar con una lista de máquinas. En lugar de ejecutar las tareas máquina a máquina, se pueden listar una serie de máquinas y Fabric visitará cada una de ellas secuencialmente y ejecutará las tareas especificadas. Esta acción puede llevarse a cabo de diferentes formas, ya sea incluyendo una lista de máquinas en una variable de entorno, pasándolas como parámetro en tiempo de ejecución o creando una serie de roles.

En Fabric un rol corresponde a una lista de máquinas, y se designa posteriormente en qué roles se van a ejecutar las diferentes tareas. Para esa lista de máquinas se asumen que cumplen un rol real, como puede ser "base de datos", de esta forma se pueden configurar una sola vez todas las máquinas donde se van a ejecutar las diferentes tareas para, más tarde, utilizar una única llamada a Fabric que ejecutaría las diferentes tareas en los destinos elegidos. En el ejemplo de la *Figura 4* pueden verse dos roles diferentes, con dos máquinas asignadas a cada uno, gracias a este mecanismo podríamos ejecutar

cualquier tarea que queramos sin tener que preocuparnos por las direcciones de las máquinas, sólo con conocer el nombre su rol.

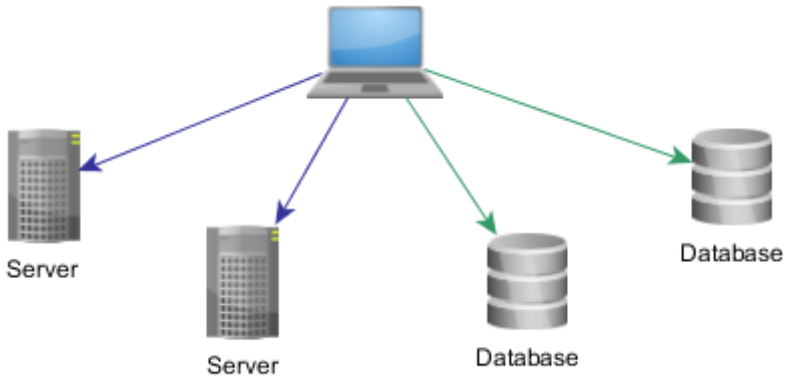

Figura 4. Ejemplo de tareas por roles

Para mantener unos buenos niveles de seguridad, Fabric permite la opción de añadir la ruta a un certificado presente en el sistema, el cual utilizará para intentar conectarse por ssh, asegurando que solo los poseedores del certificado son capaces de interactuar con la máquina [2].

2.3 Librería python-etcd

La librería python-etcd proporciona, como su nombre indica, un cliente muy ligero que permite la comunicación esencial con un clúster Etcd desde un entorno Python, sin necesidad de implementar toda la funcionalidad de comunicación independientemente [3].

Permite ejecutar de forma muy cómoda e intuitiva las acciones de escritura, lectura y borrado de claves con todas las opciones que Etcd habilita para la interacción con los servidores, como pueden ser la utilización de un ttl (*time to live*) en las claves, que permite crear una clave que expire en el tiempo especificado, o la función de *watch*, que permite observar una clave o un directorio de claves hasta que se produzca cambio y entonces devolver la respuesta, en lugar de obligar a monitorizar las claves mediante llamadas constantes al servidor. Esta última característica será muy útil para este trabajo ya que permite la utilización de la técnica de *long polling*, permitiendo emular

una metodología *push*, con lo que se reduce notablemente el estrés al que se encontrarían sometidos los servidores en el caso de utilizar *polling* normal.

A parte de estas características básicas también permite la utilización de las capacidades más complejas que ofrece Etcd, como son la elección de líder y el módulo de cerrojos, pero estas características no resultan especialmente relevantes para los objetivos propuestos en este trabajo.

2.4 Amazon Web Services (AWS)

La plataforma AWS es una colección de servicios en la nube ofrecidos por la compañía Amazon que provee una gran cantidad de funcionalidades orientada a facilitar el trabajo con la computación en la nube. Entre los servicios más populares se encuentras su plataforma de *IaaS*, Amazon EC2, y la de *PaaS*, S3, así como su servicio de correo distribuido SES.

En este proyecto el servicio que nos interesa es EC2, que nos permite desplegar servidores en la web a muy bajo coste e incluso gratis. El Amazon *Elastic Compute Cloud* (EC2) es un servicio web diseñado para facilitar a los desarrolladores un entorno de recursos escalables en la web. Permite crear y destruir máquinas muy rápidamente mediante un interfaz web, proporcionando medidas de seguridad y de privacidad desde el primer momento, así como una interfaz web con control completo sobre el estado de dichas máquinas y que permite la comunicación mediante ssh directamente desde la propia web, y asegurando que se utiliza la capacidad de computación necesaria en cada momento, ni más ni menos. Gracias a ello cumple con su principal premisa, que es ofrecer un servicio escalable a la vez que asequible.

Antes era necesario montar grandes centros de procesamiento de datos y hacer grandes despliegues físicos, que solían ser muy costosos y o bien no se aprovechaban al máximo, o bien se quedaban cortos y era necesario embarcarse en un nuevo proyecto para instalar un nuevo centro. Ahora, gracias a servicios como éste muchas más empresas pueden permitirse montar entornos distribuidos y embarcarse en la computación en la nube. Además, gracias a esto es posible crear pequeños entornos distribuidos para realizar pruebas de servicios más grandes antes de su paso a producción por un coste mínimo.

En la gráfica mostrada en la *Figura 5* se puede ver una representación del problema expuesto en el párrafo anterior. En rojo se detalla el uso real que se hace del sistema y en azul la capacidad que se solicita o que se planifica que se va a necesitar. Como se puede ver en la gráfica de la izquierda, los centros de datos tradicionales desperdician una gran cantidad de recursos si no se utilizan siempre a pleno rendimiento, y además, en el caso de que se experimente un pico de trabajo improvisto son incapaces de ofrecer una respuesta efectiva, eficaz e instantánea, causando un estado de insatisfacción en los clientes, lo que puede derivar en importante pérdida de dinero para la empresa proveedora del centro de datos. Sin embargo, la plataforma AWS ofrece un servicio elástico que es capaz de adaptarse en todo momento a las necesidades del cliente, eliminando al mismo tiempo la insatisfacción del cliente y el desperdicio de los recursos [3].

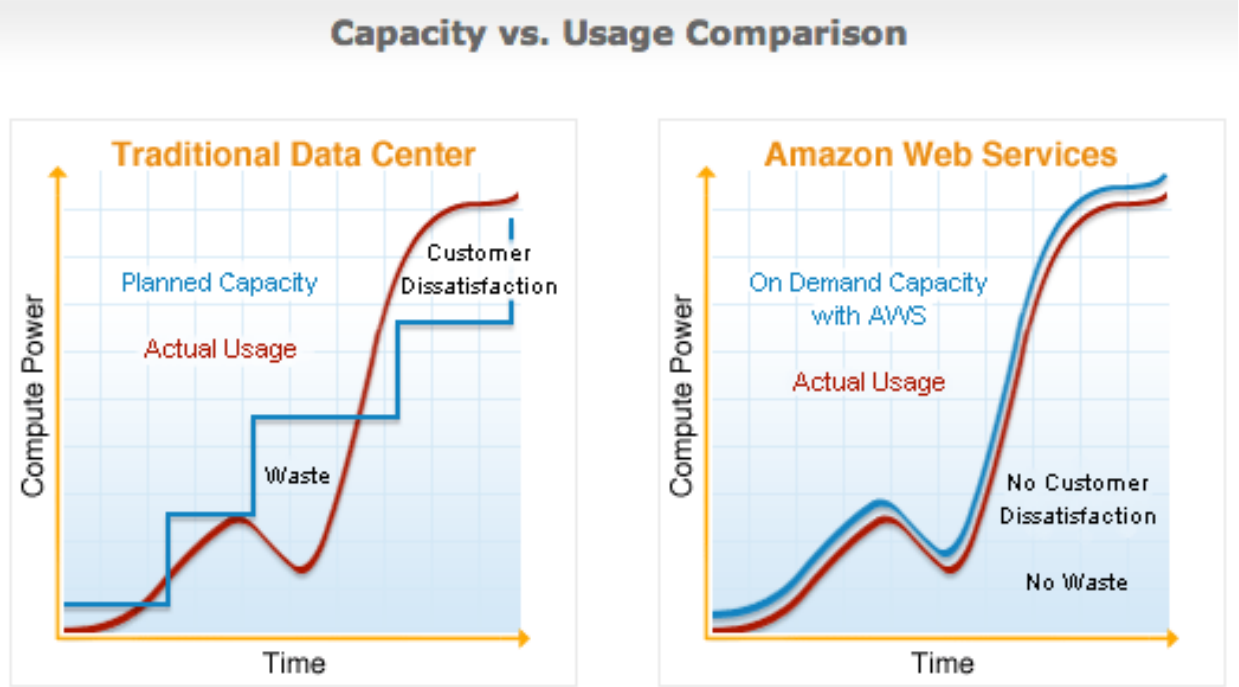

Figura 5. Comparación entre centro de datos tradicional y AWS. Fuente: Amazon Web Services

En Telefónica I+D, así como en otras grandes empresas de desarrollo de software se utilizan estos servicios para todas las necesidades de desarrollo de sistemas distribuidos, y en algunos casos también para el paso a producción de sistemas más pequeños y que requieren mayo elasticidad. También hay que tener en cuenta que a la vez que crecen los servicios solicitados también aumenta el precio, y es de vital importancia para el

buen desarrollo de la empresa mantener un equilibrio entre los servicios solicitados y su coste.

Este proyecto no se podría haber probado sin la utilización de este servicio, ya que se han podido desplegar las herramientas en entornos reales y comprobar que funcionaban correctamente y de acuerdo a lo estipulado. Para ello se desplegaron tres instancias para alojar los servidores Etcd y hasta diez clientes en otras máquinas sin ninguna funcionalidad extra, todas ellas de capacidad mínima para reducir los costes en la medida de lo posible.

3. Estado del Arte

Este estudio del estado del arte se centrará en primer lugar en dar una visión general de lo que es un sistema distribuido, para después profundizar más en qué son y cómo funcionan las herramientas de control de aplicaciones distribuidas como la que se ha desarrollado en este proyecto.

3.1 Sistemas distribuidos

Se define un sistema distribuido como un sistema en el que los componentes hardware o software de una red de computadores se comunican y coordinan mediante paso de mensajes.

No hay restricciones de distancia en esta definición, las diferentes máquinas pueden estar separadas por centímetros o encontrarse en países diferentes. De estas características se desprenden una serie de consecuencias que es muy necesario tener en cuenta a la hora de gestionar un sistema distribuido y que se comentan a continuación.

En una red de computadores lo normal es que haya varios programas funcionando concurrentemente, por lo que tanto la coordinación, como la gestión de los recursos compartidos se convierten en un punto crítico a tener en cuenta. También es importante señalar que no es posible coordinar todas las máquinas mediante un reloj absoluto, si no que se debe hacer a base de mensajes de sincronización. Por último, es necesario asumir que ningún sistema es infalible, lo que incluye tanto los sistemas independientes que componen la red, como el propio sistema distribuido, por lo tanto se añade un nivel más de complejidad a la posibilidad de fallo del sistema, ya que el fallo de una única máquina puede ser nefasto para el sistema completo, pero también puede no afectar en absoluto a su rendimiento, en caso de que se gestione correcta y eficazmente.

El principal ejemplo de sistema distribuido es Internet. Internet está formada por una gran red de computadores que se están comunicando continuamente por todas partes del mundo, a su vez, ésta red está compuesta por otras redes más pequeñas de computadores, que también forman sus propios sistemas distribuidos, así como por máquinas individuales. En la *Figura 6* podemos ver una representación a grandes rasgos de Internet.

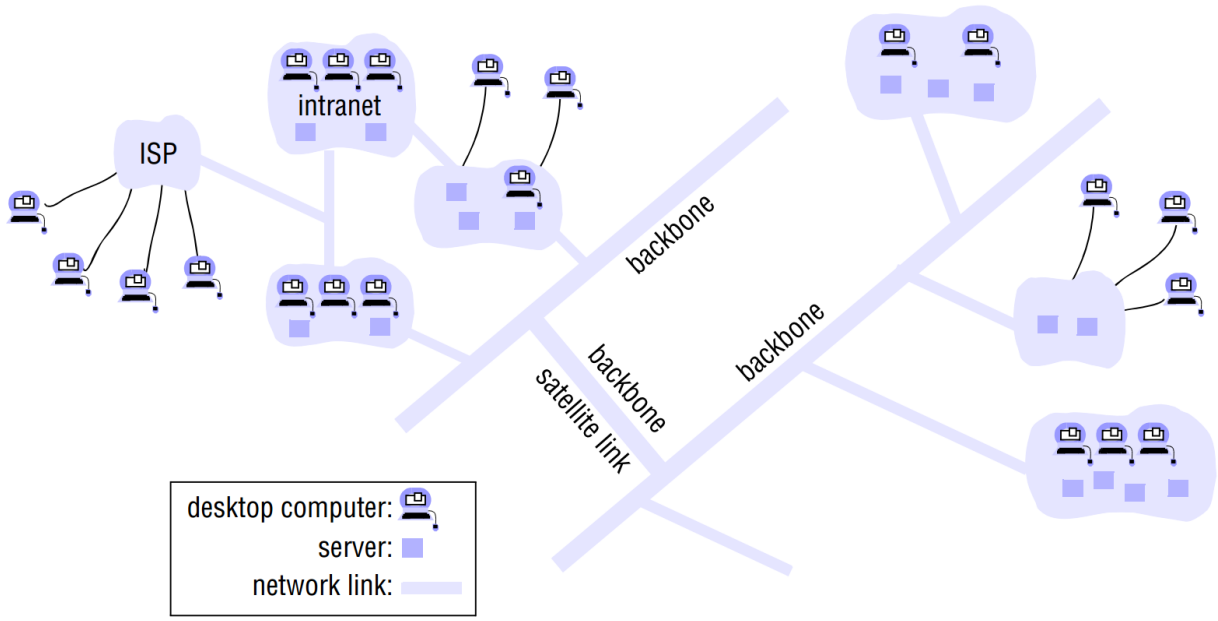

Figura 6. Esquema básico de Internet. Fuente: Distributed Sytems: concepts and design, 2005.

A día de hoy los sistemas distribuidos se encuentran en auge gracias a la gran importancia de La Nube y la enorme cantidad de redes sociales a las que la gente accede constantemente tanto desde sus ordenadores personales en casa, como desde una gran variedad de dispositivos móviles. Tanto es así, que los usuarios llegan a obviar lo que está sucediendo por detrás, y cuál es el camino que están siguiendo sus datos para llegar a todos los dispositivos de la gente con la que los comparte de modo casi instantáneo.

En cuanto a la terminología utilizada en el ámbito de los sistemas distribuidos hay dos palabras que aparecen consistentemente para referirse a cualquier sistema, las cuales vienen explicadas a continuación.

- **Servicio:** Llamamos servicio al elemento del sistema que se encarga de gestionar una serie de recursos relacionados y provee su funcionalidad a usuario y aplicaciones. Estos servicios no pueden ser accedidos de manera física, puesto que no representan recursos físicos, sino a través de la comunicación con una máquina externa.
- **Servidor:** Este término se refiere a un programa que se está ejecutando en una red de computadores y que acepta peticiones de otras máquinas para ejecutar un determinado servicio, enviándoles las respuestas apropiadas posteriormente.

Cuando hablamos de *servicios* y *servidores* nos referimos siempre, como se ha mencionado antes, a procesos que se están ejecutando en una red de computadores, y nunca a elementos físicos.

De las características mencionadas se deducen los principales retos que presenta la computación distribuida. En primer lugar, es necesario tener en cuenta que en la mayoría de las situaciones se va a tratar con sistemas heterogéneos, es decir, habrá una gran variedad de sistemas y redes de computadores que participen y se comuniquen con el sistema distribuido, además, idealmente, cada sistema estará desarrollado por personas diferentes e independientes. Además un sistema distribuido de estar abierto al cambio, debe ser capaz de incorporar nuevos programas y recursos compartidos sin alterar el funcionamiento del sistema en la medida de lo posible. El desafío anterior nos lleva a la adición de otro nuevo desafío: la escalabilidad. Un sistema ha de ser capaz de operar de manera eficaz y eficiente en diferentes tamaños, no importa si se trata de una pequeña intranet o del total de Internet. Sin embargo, a pesar de mantener ese carácter abierto, es importante también proveer un nivel de seguridad suficiente para que los clientes puedan utilizar el sistema sin ningún tipo de preocupación de que sus datos puedan estar comprometidos. También, como se mencionó al principio del capítulo, un sistema distribuido debe ser capaz en todo momento de responder ante fallos y ante situaciones bajo un alto nivel de concurrencia (miles de comunicaciones y peticiones por segundo). Por último un buen sistema distribuido será transparente al usuario a todos los niveles posibles, tanto localización, como ante fallos, como ante escalabilidad, de tal forma que un usuario no sepa qué es lo que está sucediendo dentro del sistema para que sus acciones generen respuestas correctas rápidamente.

Por supuesto, deben mantenerse la calidad de servicio mientras se cumple con los retos propuestos, para que la experiencia de usuario sea impecable [4].

3.2 Algoritmos de consenso

Cuando se habla de consenso en sistemas distribuidos se hace referencia a un conjunto de procesos que han de ponerse de acuerdo en un valor una vez alguno de los miembros propone un nuevo valor.

En el contexto de las herramientas de almacenamiento de claves distribuidas que utilizaremos para llevar a cabo la implementación de la herramienta en la que se basa este trabajo los algoritmos de consenso juegan el papel más importante. Es necesario acordar un mecanismo que sea capaz de consolidar las claves de una forma rápida, para reducir el tiempo de ejecución y de espera al mínimo, y efectiva, asegurando que el sistema es tolerante a fallos y que aún ante los mismos es capaz de responder y mantener el sistema estable.

En cada ejecución de un algoritmo de consenso se deberán cumplir las siguientes tres condiciones:

- **Finalización:** Tarde o temprano todos los procesos tienen que haber asignado un valor a su variable de decisión.
- **Acuerdo:** El valor en todos los procesos ha de ser el mismo bajo cualquier circunstancia.
- **Integridad:** Si todos los procesos correctos han elegido el mismo valor, entonces cualquier proceso correcto en el estado de decisión ha elegido ese valor.

Un algoritmo de consenso potente ha de ser capaz de solucionar todos los problemas de desacuerdo que puedan surgir, como por ejemplo el caso en el que se proponen dos o más valores diferentes para la misma variable al mismo tiempo, o el caso en el que se producen fallos de conexión entre dos segmentos diferenciados de las máquinas, lo que puede llevar a que en cada segmento se acuerden valores independientemente y en el momento de recuperar la conexión sobrevenga un nuevo problema de consenso que deberá ser solucionado cuanto antes [5].

A continuación se presentan los algoritmos de consenso Paxos y Raft. El algoritmo Paxos es uno de los más importantes y sentaría las bases para nuevos algoritmos en el futuro, como es el caso de Raft, que es el algoritmo que más se utiliza hoy en día.

3.2.1 Paxos

El algoritmo de Paxos fue descrito por primera vez por Lamport en 1989 y, si bien fue tomado a broma en un principio por muchos expertos, aún hoy está considerado como uno de los algoritmos de consenso más eficientes en un sistema de paso de mensajes.

En este algoritmo los procesos se clasifican en tres roles principales: proponentes, aceptadores y aprendices., siendo posible que un mismo proceso tenga los tres. La idea principal es que un proponente intenta ratificar un valor propuesto a base de recoger aceptaciones de la mayoría de los aceptadores, mientras los aprendices observan esa ratificación. Cuando termina, estos aprendices intentan comprobar que se ha hecho la ratificación.

Para garantizar que se progresa y que no hay problemas por empate de votos existe un rol especial, el líder. El rol de líder lo obtiene uno de los proponentes, y será el encargado de poner en marcha el algoritmo de consenso siendo el primero en recibir la orden para realizar cualquier operación.

A la hora de ejecutar el algoritmo se identifican dos fases bien diferenciadas. En cada una de estas fases el líder contacta con una mayoría y asigna un número de propuesta a cada una de las propuestas activas. Las fases del algoritmo vienen descritas a continuación:

- **Fase de preparación:** Un proponente selecciona una propuesta y envía una petición de preparación a todos los aceptadores, que han sido previamente seleccionados por el líder. Si un aceptador recibe una petición de preparación con un número mayor que cualquier otra que haya respondido antes entonces contesta con una promesa de contestar ninguna otra con número inferior, indicando el número de propuesta mayor que ha aceptado.
- **Fase de aceptación:** Si el proponente recibe una repuesta a su petición de preparación de una mayoría de aceptadores, entonces encía una petición de aceptación a todos ellos. Una vez un aceptador recibe una petición de aceptación

con un número de propuesta n, lo acepta a no ser que ya haya respondido a una petición de preparación con un número mayor que n.

Una vez se han recibido las respuestas, mediante una serie de mensajes se detecta si la mayoría se aceptadores ha aceptado la propuesta, y si es así se envía de nuevo una serie de mensajes indicando que el valor está consolidado. En el caso de que no se haya elegido un líder se puede llegar a un estado en el que no haya progreso, ya que los proponentes podrían quitarse los aceptadores entre ellos infinitamente, por eso el líder actúa como un proponente distinguido que es el único que puede generar propuestas mientras siga siendo el líder [5, 6].

3.2.2 Raft

Raft es otro algoritmo de consenso más moderno que Paxos y que está diseñado para ser más fácil de entender, pero manteniendo el rendimiento y estabilidad ofrecidos por su predecesor.

En Raft existen tres roles principales: el líder, el seguidor y el candidato (a líder). Estos roles sirven un papel similar al de los roles de Paxos, con la diferencia de que todos los procesos empiezan como seguidores y han de pasar por una fase de selección de líder antes de comenzar transmitir información.

Como se ha comentado en el párrafo anterior, existen dos etapas de funcionamiento en este algoritmo:

- **Fase de elección de líder:** En primer lugar todos los procesos son seguidores. Después de un tiempo aleatorio sin recibir ninguna comunicación desde un líder uno de ellos se presenta como candidato y empieza a solicitar votos a todos los seguidores, que le contestan con su voto. Una vez consigue una mayoría de votos se consolida como el líder. En el caso de que más de un seguidor se convierta en candidato al mismo tiempo y consiga el mismo número de votos se volverá a empezar un tiempo aleatorio, asegurando que no vuelven a solicitar votos al mismo tiempo, de esta manera se asegura que uno de ellos conseguirá el liderazgo y el otro lo aceptará y volverá a convertirse en seguidor. A continuación, en la *Figura 7*, se muestran unas imágenes que ilustran el proceso básico de elección de líder, extraídas de una presentación de Ben Johnson.

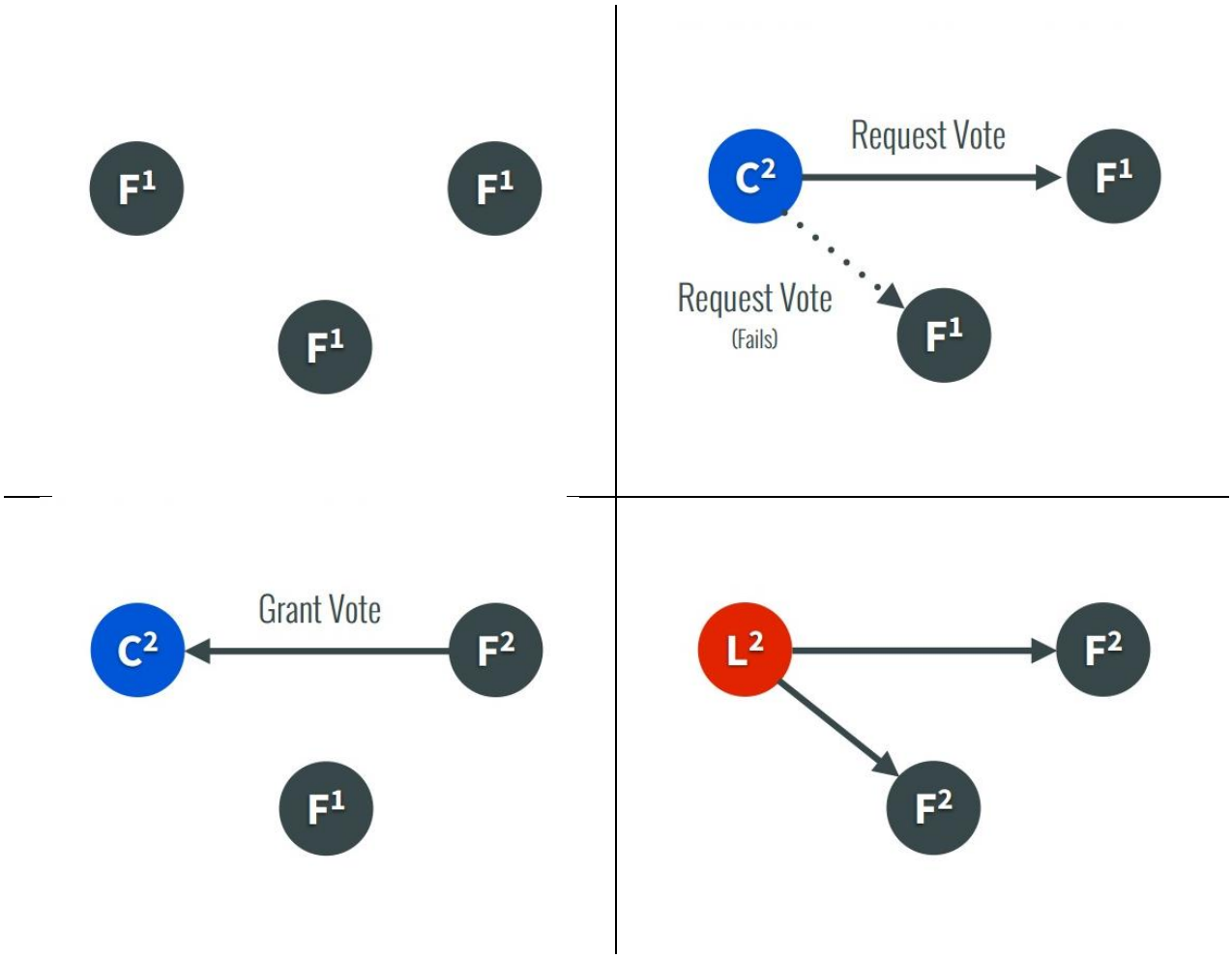

Figura 7. Figuras ilustrando el proceso de elección de líder. Fuente: Ben Johnson, 2013.

- **Fase de replicación de registros:** Cada proceso participante tiene un registro en el que almacena todos los valores. A la hora de añadir un nuevo valor al registro, este valor será introducido directamente en el líder y él mismo enviará una comunicación a todos los seguidores indicando el nuevo valor. Cuando los seguidores hayan recibido la comunicación responderán al líder con un mensaje de aceptación, dando a entender que han recibido la información y que están conformes. Cuando la el líder recibe la mayoría de las respuestas, el líder envía un nuevo mensaje a todos los seguidores indicando que ese valor está consolidado. Como en el apartado anterior, en la *Figura 8* se muestran otro fragmento de la presentación que ilustra este proceso.

Figura 1. Figuras ilustrando el proceso de replicación. Fuente: Ben Johnson, 2013.

Estas fases se suceden siempre que es necesario en el sistema, asegurando que los valores se consolidan con valores estables. En el caso de que se produzca algún error de conexión que divida el clúster, las fases se repetirán en las secciones divididas. Es importante tener en cuenta que en esta situación, a la hora de elegir un nuevo líder seguirá siendo necesario que el candidato reciba la mayoría de votos del total de procesos que componen el sistema [7, 8].

3.3 Tecnologías para configuración de aplicaciones distribuidas

Existe una gran variedad de tecnologías en el campo de la configuración de aplicaciones distribuidas, sin embargo la mayoría de ellas se encuentran integradas en suites que ofrecen una mayor funcionalidad, generalmente de despliegue de aplicaciones. En general estas herramientas suelen estar muy completas y presentar muchas características, algunas útiles y otras no tanto, sin embargo el objetivo de este trabajo es desarrollar

una herramienta sencilla y ligera que sea capaz de realizar una pequeña parte de ese trabajo, en concreto manejar la configuración de una serie de máquinas desplegadas que ejecuten aplicaciones distribuidas.

Entre las herramientas existentes que se limitan a realizar esta tarea podemos destacar las siguientes:

3.3.1 Apache Zookeeper

Zookeeper es un sistema de coordinación de aplicaciones distribuidas de alto rendimiento. Está diseñado para que sea fácil de programar, pero manteniendo una buena base para construir aplicaciones distribuidas. Está diseñado para ser utilizado con sistemas Java, pero también permite utilizar implementaciones en el lenguaje C.

Los principales objetivos de Zookeeper son los siguientes:

- La coordinación de procesos ha de ser **simple.**
- El sistema se puede **replicar** entre varios servidores, para minimizar el impacto de los fallos.
- Las actualizaciones están **ordenadas** explícitamente mediante un número de versión.
- Es **rápido** en entornos donde predominan las lecturas.

La información se almacena siguiendo una estructura de datos en forma de árbol, similar a un sistema de archivos, en la que unos nodos especiales, denominados znodos, almacenan información de versión, localización, etc., a parte de la propia información que se quiera almacenar en dicho znodo. Podemos ver un ejemplo de esta estructura en la *Figura 9,* extraída de la página oficial de Zookeeper.

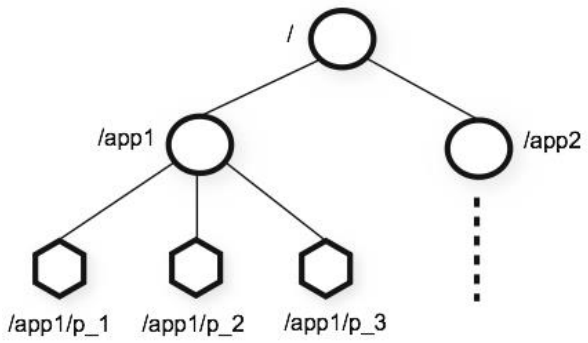

Figura 2. Esquema de nodos de Zookeeper. Fuente: http://zookeeper.apache.org/, 2013.

La API es muy básica y sencilla de utilizar, lo que cumple con la propuesta de simplificar el trabajo del desarrollador que se hacía anteriormente, aunque parece estar algo limitada en cuanto a características más específicas, como puede ser la monitorización en tiempo real de la actividad del sistema.

Zookeeper garantiza cualidades muy interesantes en cuanto al almacenamiento de datos, como son la consistencia, atomicidad en las operaciones, una única imagen del sistema completo (es decir, no importa a qué servidor se conecten las máquinas, siempre verán lo mismo), confiabilidad en las actualizaciones y que los clientes van a estar actualizados en todo momento. En ningún momento se menciona ningún objetivo de seguridad, por lo que probablemente debería ser implementado aparte.

En la web de Zookeeper se ofrecen varias gráficas de rendimiento bastante prometedoras, pero debido a las dimensiones que se manejan para comparativas de rendimiento sustanciales no se han podido realizar pruebas especialmente reveladoras en local.

Zookeeper es una tecnología bastante antigua en comparación con las que se comentarán a continuación, sin embargo sentó la base de los sistemas de coordinación y configuración de aplicaciones distribuidas. Carece de algunas características que se consideran importantes como son la seguridad, y las limitaciones en cuanto a los lenguajes aceptados, de modo que probablemente no sea la mejor opción en el contexto de este TFG [9].

3.3.2 Doozerd

Doozerd promete literalmente "un sistema capaz de almacenar pequeñas cantidades de datos extremadamente importantes manteniendo a la vez un nivel completo de consistencia y una alta disponibilidad". Está pensado para sistemas que requieren realizar muchas lecturas y muy pocas escrituras, como pueden ser servicios de nombres o datos de configuración de sistemas (que es la característica que más atrae desde el punto de vista de este proyecto).

El total de los datos son almacenados en memoria, lo que permite que sean tratados con mayor rapidez, a cambio de un aumento del uso de memoria. Además se indica que no

se proveen herramientas para la escritura en disco ni *backup*, parte que deberá ser implementada por separado.

Los datos se organizan siguiendo una estructura de ficheros en forma de árbol, muy similar al sistema de ficheros utilizado en Unix. Cada directorio contiene una lista de nombres que se corresponde con otros directorios o archivos, siendo estos nombrados, como se mencionó antes, de la misma manera que en Unix, utilizando "/" para separar los diferentes directorios hasta el archivo final. Los usuarios están limitados a realizar escrituras o lecturas completas del fichero, nunca a editar partes del mismo. Además, para asegurar la consistencia el servidor guarda una serie de revisiones. Estas revisiones consisten en versiones completas de los datos cada vez que se actualizan, es decir, cada vez que se produce un cambio en cualquiera de las claves se realiza una copia completa de todos los datos almacenados, cambiando su número de versión, de esta manera se puede volver fácilmente a versiones anteriores de los datos. Estas revisiones no se mantienen eternamente, sino durante un periodo de tiempo configurable.

Uno de los puntos fuertes, según la documentación oficial es la ya mencionada alta disponibilidad así como la tolerancia ante fallos. Se especifica que es capaz de reconocer nodos muertos en el clúster y eliminarlos por completo del sistema tras un tiempo de *timeout* personalizable, permitiendo redirigir a los clientes a otros nodos que sigan funcionando para mantener la estabilidad del clúster sin que estos fallos afecten al rendimiento de los clientes en ningún momento, siempre y cuando alguno de los nodos se mantenga en funcionamiento.

El sistema Doozerd también provee una vista web de los datos almacenados en el clúster. Esta web está disponible accediendo a cualquiera de las máquinas que componen el clúster y permite observar los datos con una interfaz bastante amigable (ver *Figura 10*). Es una interfaz de sólo lectura, no permite editar ni añadir datos de ninguna manera, pero puede ser utilizada para obtener rápidamente una idea completa del estado del servidor de configuración en cualquier momento [10].

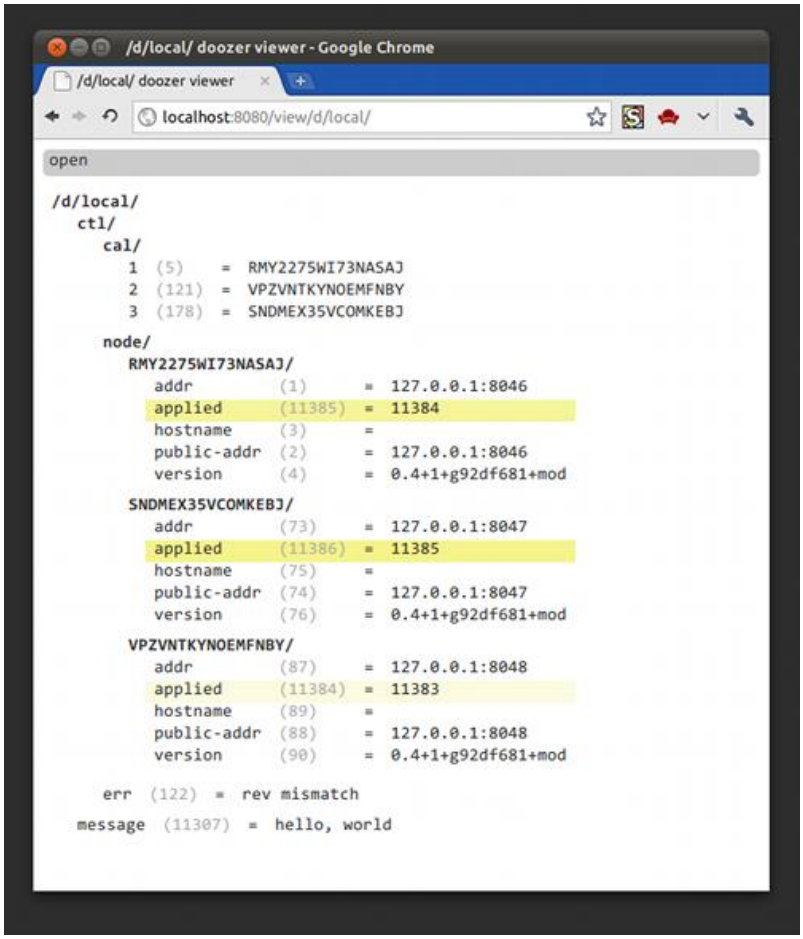

Figura 3. Vista web de Doozerd. Fuente: https://github.com/ha/doozerd, 2013.

Por último, cabe destacar algunos de los principales fallos que se han encontrado durante la investigación de esta herramienta. En primer lugar es importante tener en cuenta la escasa documentación. Ésta se encuentra limitada a los pocos párrafos incompletos que se pueden encontrar escritos en su repositorio de Github, así como a algunos otros comentarios publicados en diversas partes de internet, además de una exploración superficial del propio código de la herramienta. En segundo lugar la ausencia de implementaciones predefinidas de seguridad hace necesaria la inclusión de técnicas de seguridad externas de las que tendría que ocuparse el desarrollador o el administrador de los servidores y complica los despliegues. Finalmente, y como punto más importante, la versión que se encuentra publicada en el repositorio oficial falla al compilar, es posible que haya habido alguna modificación al código de las dependencias externas de la herramienta, pero no se puede compilar esa versión, y por tanto no se ha podido arrancar ni hacer pruebas con ella.

3.3.3 Etcd

Etcd es un sistema de almacenamiento de claves con alta disponibilidad, creado por CoreOS para la gestión de sus propios clústeres con una gran capacidad para recuperarse ante fallos, especialmente del líder de clúster especialmente. Etcd ha sido especialmente diseñado para configuración compartida de aplicaciones distribuidas y descubrimiento de servicios. Es un sistema relativamente actual y está basado en los dos mencionados en los apartados anteriores, pero con unos objetivos muy claros. Estos objetivos son:

- **Sencillez:** Es accesible por el usuario de manera muy simple mediante una API HTTP que devuelve resultados de tipo JSON y puede ser accedida sin la necesidad de un programa intermediario, utilizando la herramienta curl, presente en la mayoría de sistemas Unix.
- **Seguridad:** Permite la inclusión de medidas de autenticación mediante el uso de certificados SSL.
- **Rapidez:** Se han llegado a registrar en diversas pruebas miles de escrituras por segundo en instancias individuales del sistema.
- **Confiabilidad:** Es capaz de mantener los datos almacenados de una manera fiable utilizando el algoritmo de consenso Raft.

Como se ha comentado anteriormente, el algoritmo Raft especifica una arquitectura con un líder y una serie de seguidores. De este modo las escrituras se pueden realizar en cualquier miembro del clúster, este las redirige al líder, y desde el líder se redistribuyen a todos los demás seguidores, como se muestra en la *Figura 11.* Según este mecanismo una escritura se considera consolidada una vez la mayoría de los servidores hayan recibido la información.

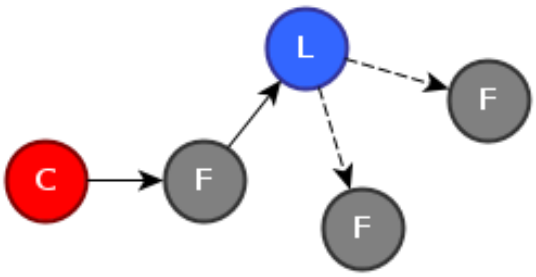

Figura 4. Diagrama de comunicación en Etcd

Otra característica muy importante y atractiva de Etcd es que viene con un sistema integrado de descubrimiento de servicios, y ofrece un sistema online donde puedes registrar tu propio clúster de servidores para que se conecten unos a otros automáticamente al arrancar y no sea necesario mantener un registro aparte de los mismos y actualizarlo cada vez que se quieran arrancar nuevas máquinas. Esto facilita mucho el trabajo del desarrollador, especialmente a la hora de añadir alta disponibilidad al servicio que utilice Etcd, ya que se recomienda tener al menos tres máquinas funcionando, y siempre en número impares para aprovechar sus capacidades al máximo. La razón de que se recomiende un número impar de máquinas viene dada por la forma en que funciona el algoritmo Raft. Al necesitarse una mayoría para consolidar una clave, un clúster de ocho máquinas, por ejemplo, necesitaría el consenso de cinco máquinas, siendo el mismo caso para un clúster de nueve máquinas, sin embargo, el clúster de nueve máquinas permite que fallen cuatro de las máquinas sin afectar al resultado, mientras que el de ocho solo permitiría el fallo de tres máquinas. Como podemos observar en la *Figura 12*, el líder (L) ha recibido una nueva clave del cliente (C) y se la ha mandado a sus ocho seguidores. Sólo cuatro de los seguidores responden, lo que conforma una mayoría de cinco máquinas, por lo que la clave se consolida, y las otras cuatro máquinas se actualizarán cuando recuperen la conexión.

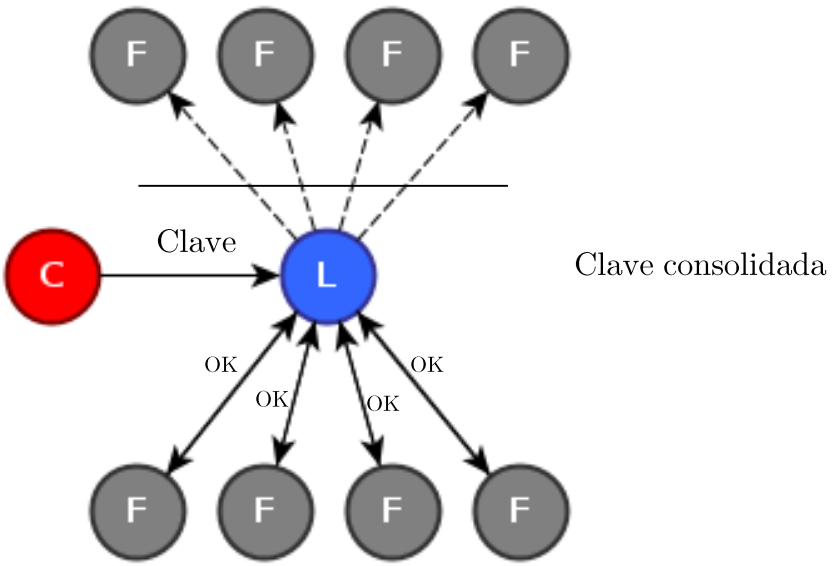

Figura 5. Ejemplo de clúster de nueve máquinas aceptando una clave

Etcd se ha desarrollado por completo utilizando el lenguaje Go. Este lenguaje ha sido desarrollado por Google y tiene una sintaxis similar a la del lenguaje C. Fue desarrollado con el principal objetivo de ser un lenguaje para sistemas concurrentes y es utilizado generalmente para la programación de servidores que van a estar expuestos a un gran número de peticiones y van a tener una gran carga de trabajo concurrente.

Uno de los principales defectos de Etcd radica en que su latencia se ve reducida a la del miembro más lento del clúster, y esto puede llevar a problemas de inestabilidad en determinados escenarios, sin embargo este problema se puede evitar ajustando debidamente las máquinas para minimizar dichos problemas [11].

Finalmente se realizaron una serie de pruebas muy básicas del sistema, que consistieron simplemente en instalar un pequeño clúster de servidores y utilizar la herramienta curl para escribir algunas claves y comprobar que todo funciona debidamente y como se esperaba.

Esta sería finalmente la herramienta escogida para utilizar como herramienta de gestión de la configuración distribuida. Una de las principales razones para hacer esta elección es que se trata de una herramienta que se encuentra actualmente en continuo desarrollo, por lo que se espera que siga mejorando con el paso del tiempo. También se ha tenido en cuenta el hecho de que provee opciones para incluir medidas de seguridad en los intercambios de datos de manera sencilla y que las instancias de servidor necesarias para su funcionamiento son muy ligeras, lo que permite un despliegue rápido. Y, en especial, que es muy sencillo trabajar con ella, sin necesitar el apoyo de otras herramientas externas.

4. Desarrollo

En este capítulo se recoge todo el proceso de creación de la herramienta, desde los orígenes de la idea, hasta la documentación del producto final, pasando por todas las etapas del diseño. Se detallan también aspectos relevantes del desarrollo efectuado tales como el ciclo de vida seguido, las pruebas realizadas, el entorno de gestión de configuración y tickets utilizado, y la licencia del producto.

4.1 Introducción

La idea de este proyecto surgió a partir de una prueba de concepto que se hizo en Laboratorio de Innovación Abierta de Telefónica Digital y la UPM. La prueba de concepto consistía en demostrar que se podía crear una pequeña herramienta capaz de encargarse del despliegue e instalación de máquinas remotas completamente desde una máquina local, facilitando el trabajo del desarrollador y de los administradores al eliminar la necesidad de conectarse manualmente a cada una de ellas.

En la primera versión se empezaron a investigar las capacidades de Fabric y se llegó a desarrollar un sistema capaz de copiar archivos a una máquina remota, ejecutar algunas instrucciones sencillas y abrir una consola desde la que se podían ejecutar mandatos como si se estuviera haciendo en la propia máquina local. Más adelante se intentó añadir la funcionalidad de provisionar nuevas máquinas utilizando la API de Amazon Web Services, sin embargo hubo algunos problemas que ralentizaron bastante el desarrollo, consiguiéndose únicamente configurar máquinas individuales, en lugar de listas de máquinas. Este proyecto se dejó de lado por un tiempo debido a otra serie de asuntos de más urgencia en el laboratorio, sin embargo surgieron una serie de temas

interesantes a tratar para futuras versiones, uno de los cuales ha servido como tema principal para este proyecto.

Por un lado sería necesario mejorar las capacidades de despliegue de la herramienta, gestionando de una manera más eficaz las capacidades ofrecidas por AWS, automatizando todos los procesos de despliegue de máquinas e instalación de dependencias en la medida de lo posible. Esto sirvió como tema principal para otro trabajo de fin de grado que se ha desarrollado de forma paralela por otra persona en el mismo laboratorio.

Por otro lado surgió la necesidad de utilizar algún tipo de herramienta que permitiera coordinar la configuración de las diferentes máquinas, también de forma automatizada. Gracias a esta herramienta se podrían configurar todas las máquinas desde local sin preocuparse de nada en los despliegues, ya que la configuración dejaría de ser responsabilidad del desarrollador o administrador encargado del despliegue y pasaría a ser recibida desde un lugar centralizado. Esta idea es la que dio pie al proyecto desarrollado en este trabajo.

Lo más lógico era integrar las dos herramientas para obtener un sistema completo capaz de manejar todas las facetas de un despliegue, sin embargo, apostando por la sencillez, se decidió separar la funcionalidad en dos herramientas completamente independientes, que pudieran ser utilizadas la una sin la otra, para más adelante pensar en una integración más completa que pueda sacar el máximo partido a la combinación de ambas.

Una vez tomadas estas decisiones se procedió con el diseño y la implementación de una herramienta capaz de preparar una forma de almacenar una configuración compartida para todas las máquinas que fuera accesible desde cualquier lugar y en cualquier momento estando siempre completamente actualizada.

4.2 Diseño de la solución

La etapa de diseño fue relativamente larga y sufrió varios cambios de opinión sobre la dirección que se debía tomar.

4.2.1 Primera aproximación

En un principio, cuando se estaba desarrollando la prueba de concepto mencionada en la introducción de este capítulo, se pretendía utilizar la propia herramienta (desde ahora denominada CLI, por su estilo de "*command line interface*") para mandar la configuración a las máquinas directamente y de forma manual cada vez que hubiera que realizar un cambio. Ni siquiera se consideraba la opción de tener una herramienta aparte para realizar las tareas de configuración. En la *Figura 13* se puede observar un esquema de esta aproximación.

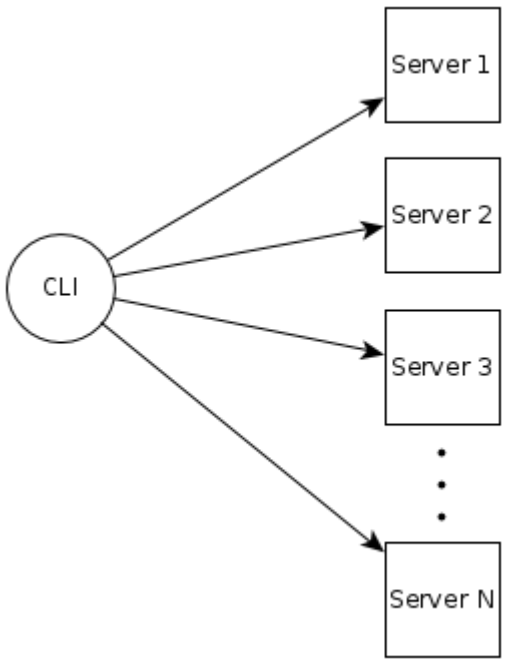

Figura 6. Diagrama de la primera aproximación

Problemas

Como se puede comprender, esta aproximación tiene unas limitaciones muy grandes:

- **Es necesaria la interacción de la herramienta CLI:** No se trata de un proceso automatizado, por lo que cada actualización de alguna variable de configuración requiere la utilización de la herramienta de despliegue, que tal y como estaba concebida suponía que una persona ejecutara el mandato manualmente.
- **No es posible asegurar la fiabilidad:** En caso de que la subida del fichero fallara por cualquier motivo sería necesario repetir la operación, y durante este tiempo

las máquinas de la red distribuida estarían operando con configuraciones diferentes, lo que podría causar inestabilidades en el funcionamiento de la aplicación.

- **Dificulta la escalabilidad:** Si se necesitara distribuir la nueva configuración a un número elevado de máquinas tanto los tiempos de actualización como los fallos podrían dispararse.

Conclusión

En resumen, no se consideró como una buena práctica de diseño. Podría funcionar en entornos pequeños, pero de cara a montar un sistema de un tamaño más considerable no supone una buena solución.

En este momento se decidió que sería mejor crear una herramienta independiente, que fuera capaz de encargarse de todas las tareas de gestión de la configuración distribuida, de tal modo que el uso de la herramienta CLI no fuera obligatorio para el usuario, que tal vez tuviera ya sus máquinas montadas, pero le interesara incluir un sistema distribuido que ayudara a mantener la configuración lista. Esto dio lugar a la segunda aproximación.

4.2.2 Segunda aproximación

Para una segunda aproximación se decidió, como se ha mencionado antes desarrollar una herramienta completamente independiente, capaz de realizar todas las tareas relacionadas con la configuración.

Para solucionar los problemas que se habían descubierto en la primera aproximación de una manera segura y relativamente rápida se decidió no reinventar la rueda y hacer una gran investigación sobre las diferentes tecnologías existentes que se dedican a realizar tareas similares. Lo primero fue descartar grandes herramientas que presenta una funcionalidad mucho mayor de lo que se intenta proveer, aumentando la complejidad a un grado completamente innecesario, ejemplos de estas herramientas son Chef o Puppet. Se buscaron herramientas más ligeras pero que sigan proporcionando estabilidad, fiabilidad, resistencia a fallos y escalabilidad. Esta investigación se encuentra bien detallada en el *Capítulo 3*, en el apartado *Tecnologías examinadas*, donde se especifica que finalmente se eligió la herramienta Etcd.

Etcd proporciona la capacidad de mantener un clúster de servidores con alta disponibilidad que se comunican entre ellos y mantienen una única imagen actualizada del almacén de claves en cada momento.

Arquitectura

La idea principal de esta aproximación era crear un ligero cliente local con capacidad para instalar en unas máquinas remotas un clúster de servidores Etcd y para escribir claves en ellos y leerlas después.

La herramienta de consola local, a la que a partir de ahora denominaremos manager, será la encargada de toda la comunicación con el servidor. Siempre que haya que actualizar las claves se hará a través de este manager, siendo innecesario ningún tipo de conexión directa con los servidores que conforman el clúster.

Una vez las claves están consolidadas en el servidor la intención era utilizar un modelo *push* para trasladarlas a los servidores de la aplicación distribuida. De esta forma cada servidor Etcd tendría la responsabilidad de comunicarse con todas las máquinas que tenga asignadas y transmitirles la nueva configuración, para mantenerlas completamente actualizadas en todo momento. En la *Figura 14* se muestra un diagrama que ilustra la arquitectura de esta aproximación.

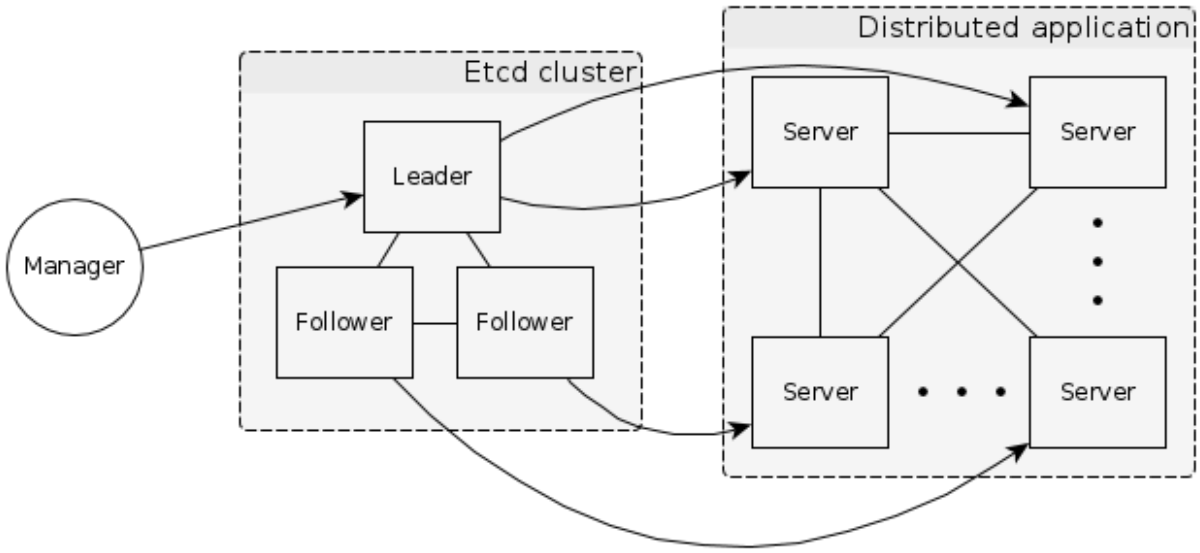

Figura 7. Diagrama de la segunda aproximación

Problemas

Esta aproximación, a pesar de parecer la solución ideal por su simpleza y efectividad presenta algunos problemas, especialmente por limitaciones de la herramienta Etcd. A continuación se presentan los problemas en más detalles:

- **El sistema Etcd no provee un sistema *push*:** Sería necesario el diseño y la implementación de un servicio capaz de monitorizar los servidores Etcd y realizar las acciones de *push*. Esto añadiría una capa de complejidad a los servidores, y haría necesario registrar todas y cada una de las máquinas asociadas a la aplicación distribuida.
- **La aplicación no se puede comunicar con el clúster Etcd:** Es posible que la aplicación desee modificar alguno de los valores almacenados en el sistema de claves para que lo puedan ver el resto de las máquinas, opción que esta aproximación *push* no ofrece.

Conclusión

A pesar de los problemas presentados, la herramienta Etcd había demostrado ser muy potente y cumplir con la funcionalidad necesaria. Tras contrastar las ventajas y desventajas de este diseño se decidió que no era buena idea violar los principios de trabajo de la Etcd y que sería mejor mantenerla lo más intacta posible, respetando su forma de trabajo.

Con estas limitaciones en mente y la intención de implementar una herramienta que permitiera la ampliación de su funcionalidad en un futuro, surgió la tercera aproximación, que acabaría siendo la definitiva.

4.2.3 Tercera aproximación, la definitiva

En la solución definitiva se decidió, como ya se ha mencionado, mantener la estructura que incluye el clúster Etcd, pero además implementar un cliente capaz de comunicarse con el servidor mediante llamadas de tipo *pull*, que funcionara como un servicio instalado en cada una de las máquinas que forman parte de la aplicación distribuida.

Este cliente en principio cumplirá únicamente la función de mantener un fichero de configuración actualizado en todo momento con toda la información de configuración incluida en el clúster de Etcd, que la aplicación estará encargada de monitorizar para

mantenerse al día con el servidor. En un futuro, como se explicará en el Capítulo 6, Líneas futuras, el cliente sería capaz de comportarse de manera interactiva con la aplicación y permitir a esta escribir sus propias claves en el servidor de configuración, incrementando la cooperación entre máquinas.

Arquitectura

La arquitectura de esta última aproximación es muy similar a la arquitectura de la segunda aproximación, sin embargo en este modelo todo podrá ser controlado desde el manager.

En este diseño el manager tendrá la capacidad de instalar tanto el clúster de servidores Etcd como los clientes de todas las máquinas pertenecientes a la aplicación distribuida, además será capaz no sólo de escribir claves directamente en el clúster como antes, sino también de detener y reiniciar los clientes. A continuación, en la *Figura 15*, se muestra una ilustración que representa la arquitectura diseñada para esta nueva etapa.

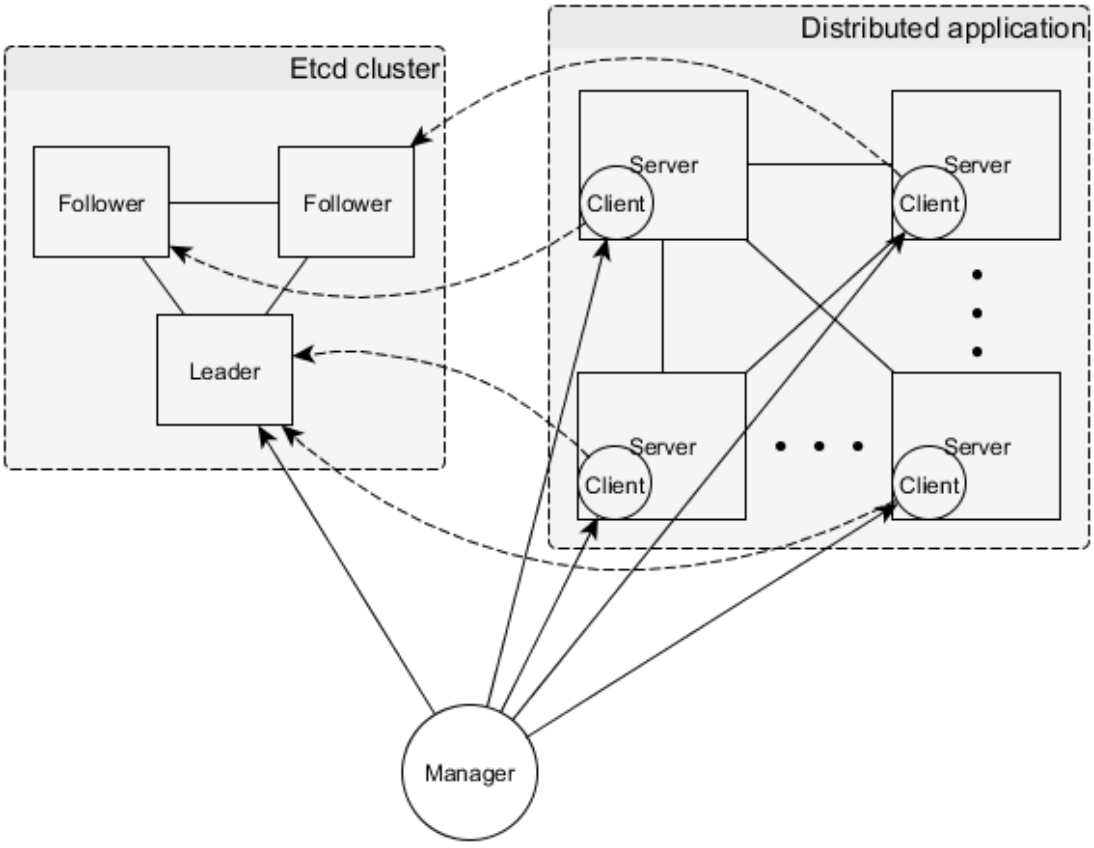

Figura 8. Diagrama de la tercera aproximación

En el diagrama se han marcado con una línea sólida las interacciones directas del manager y con una línea de puntos la llamadas *pull* que utilizarán los clientes implantados en las máquinas de la aplicación distribuida.

Las llamadas *pull* de los clientes son actividades de observación utilizando la técnica del *long polling*. Esta técnica es una variación del *polling* tradicional que permite emular un mecanismo de tipo *push* realizando peticiones de tipo *polling* normales, solo que con una frecuencia mucho menor. Si el servidor no tiene información nueva que devolver al cliente, en lugar de devolver una respuesta vacía, guarda la nueva petición y espera o bien a tener información disponible, o bien a que pasa un período de tiempo determinado. Una vez se cumple alguna de las dos condiciones el servidor envía una respuesta y el cliente la trata y vuelve a realizar otra petición de tipo *pull*, para mantener siempre una llamada activa en el servidor [12].

Los clientes integrados en la aplicación actuarán como servicios, con la capacidad de iniciarse con la máquina y reiniciarse en caso de fallo, actuando de manera completamente transparente para ella.

Conclusión

Esta solución pareció ser la más acertada para cumplir con los objetivos propuestos, además de solventar los problemas encontrados en la segunda aproximación, además de no incluir, aparentemente, nuevos problemas de diseño para el futuro. Con esta última aproximación conseguimos liberar a la aplicación distribuida de la mayor parte de la responsabilidad, teniendo sólo que monitorizar los cambios en un archivo de configuración, actividad que probablemente ya realizara de todas formas, y también liberar de responsabilidad a los miembros del clúster de configuración, gracias a lo cual se puede mantener una instalación limpia de Etcd sin necesidad de añadir modificaciones.

Además habilita la interacción directa con los clientes, permitiendo arrancarlos, pararlos o reiniciarlos, y en futuro se podrá añadir más funcionalidad de forma sencilla si así se requiere, ya que todo será controlado en local desde el manager, y no será necesario en ningún momento conectarse a las máquinas remotas.

A continuación se describe la herramienta y sus funcionalidades en más detalle.

4.3 Documentación del producto final

La herramienta desarrollada en este proyecto de fin de grado ha terminado por recibir el nombre "etcd-manager" y ha sido diseñada para ser capaz de controlar la configuración de una aplicación distribuida de forma centralizada desde un entorno local. La configuración se encontrará disponible en todo momento gracias a la utilización de un clúster de máquinas que utilizará la tecnología de almacenamiento de claves distribuidas Etcd.

4.3.1 Componentes

El etcd-manager consta de tres partes principales, como se ha indicado en la tercera etapa de diseño:

- **Manager:** Es el gestor principal y la única parte de la herramienta con la que el usuario va a interactuar durante el uso de la herramienta. Se trata de una herramienta de consola y es la encargada de realizar todas las posibles funciones que se detallarán más adelante, siendo capaz de comunicarse con las otras dos partes de forma imperativa y directa.
- **Clúster Etcd:** Es un servidor o clúster de servidores que preservan la funcionalidad completa de la tecnología Etcd intacta. Aquí es donde se almacenará toda la información de configuración de la aplicación distribuida, manteniéndola con la alta disponibilidad que permite Etcd.
- **Cliente ligero:** Se trata de un pequeño cliente que estará implantado en todas las máquinas miembros de la aplicación y se encargará de mantener un fichero actualizado con toda la información disponible en el clúster Etcd. Como ya se detalló en el apartado de diseño, este cliente utiliza un modelo *long polling* para emular un método *push* del servidor.

4.3.2 Instalación

La instalación de la herramienta es muy sencilla. Basta con descargar la herramienta, descomprimirla e instalar las dependencias contenidas en el archivo *requirements.txt* utilizando pip de la siguiente forma:

```
$> pip install -r requirements.txt
```

4.3.3 Configuración

Una vez instalada la herramienta se deberá o bien incluir en el archivo de configuración *default_config.json* o en un archivo de configuración propio que se deberá indicar en la llamada al programa, la información que la herramienta necesita para su ejecución en formato json. Esta información se divide en dos partes: la información de los servidores y la información de los clientes.

```
{
    "client": {
        "key_filenames": [
            "/path/to/certificate.pem"
        ],
        "hosts": [
            {
                "username": "name",
                "url": "url"
            },
            {
                "username": "name",
                "url": "url"
            }
        ],
        "config_path": "/path/to/config.json "
    },
    "server": {
        "key_filenames": [
            "/path/to/certificate.pem"
        ],
        "hosts": [
            {
                "username": "name",
                "url": "url"
            }
        ],
        "discovery_token": "token"
    }
}
```

Figura 9. Ejemplo de fichero de configuración

Ambas contienen unos campos comunes que son:

- **hosts**: Contiene una lista de *hosts*. Cada *host* representa una de las máquinas en las que se van a ejecutar las tareas que implican máquinas concretas, como por ejemplo las tareas de instalación. Cada uno de los elementos de esta lista ha de

contener una *url* o dirección IP mediante la que acceder a la máquina así como un nombre de usuario (*username)* desde el que se ejecutarán las acciones (tiene que ser un usuario existente en la máquina y con permisos de superusuario).

- **key_filenames:** Contiene una lista con las rutas de los archivos de certificado que se usarán para realizar las conexiones ssh a las máquinas especificadas en la variable *hosts*. La herramienta Fabric utiliza la lista de archivos para acceder a todas las máquinas, por lo que no es necesario especificar a qué máquina corresponde cada archivo.

Además de esos campos comunes en las dos secciones, cada una tiene un campo más específico de la misma:

- **config_path (cliente):** especifica la ruta del archivo donde se almacenará la configuración descargada por los clientes.
- **discovery_token (server):** especifica un *token* que los servidores Etcd utilizarán para encontrar a los demás miembros del clúster y conectarse a ellos. Esta variable no es necesario rellenarla manualmente en el caso de que queramos obtener un nuevo *token*, se ha creado una función para ello, que estará especificada en el apartado Funciones, más adelante.

4.3.4 Ejecución

La herramienta ya está lista para su utilización mediante la ejecución por consola del mandato:

```
$> etcd-manager [-c CONFIG_FILE] action [arguments]...
```

El *CONFIG-FILE* es la ruta del archivo de configuración que queremos utilizar, y es un parámetro opcional. El archivo provisto debe cumplir la especificación descrita en el apartado anterior. La *action* es la tarea que queremos realizar en las máquinas especificadas en archivo de configuración. Los *arguments* son posibles argumentos que se les pueden dar a las *action*, como por ejemplo el nombre y el valor de la clave que queremos escribir en el servidor. Las *action* y los *arguments* vienen explicados en la siguiente sección.

4.3.5 Funciones

En este apartado se especifican las funciones principales de la herramienta y la forma correcta de utilizarlas.

Funciones para los servidores

Existen funciones para instalar, iniciar, parar y reiniciar máquinas del clúster Etcd, para leer, escribir y borrar claves del clúster y una última función para obtener un *discovery_token* que permite la interacción entre máquinas del clúster.

Action	Arguments	Descripción
install_server	[go_path, etcd_path]	Instala una instancia de etcd en cada uno de los *hosts* especificados en la sección *server* del fichero de configuración. Estas funcionarán como servicios en la máquina destino. Los dos argumentos opcionales *go_path* y *etcd_path* son las rutas a versiones de Go y de Etcd diferentes a las que vienen por defecto, por si se quieren utilizar versiones diferentes. Es necesario proveer un *discovery_token* o solicitar uno mediante get_discovery_token para poder utilizar esta acción
start_server	-	Inicia todos los servidores incluidos en la lista
stop_server	-	Detiene todos los servidores incluidos en la lista
restart_server	-	Reinicia todos los servidores incluidos en la lista
read	key	Lee una clave (*key*) del clúster Etcd
write	key, value	Escribe en la clave *key* del clúster Etcd el valor *value*, sobrescribiéndola si ya existe
delete	key	Elimina una clave (*key*) del clúster Etcd
get_discovery_token	-	Obtiene un nuevo *discovery_token* si no se ha especificado y lo escribe en el fichero de configuración

Funciones para los clientes

Para los clientes se presenta una funcionalidad algo más limitada debido a su naturaleza. Existen funciones para instalar, iniciar, parar y reiniciar los clientes.

Action	Arguments	Descripción
install_client	-	Instala un cliente en cada uno de los *hosts* especificados en la sección *client* del fichero de configuración. Estos funcionarán como servicios en la máquina destino.
start_client	-	Inicia todos los clientes incluidos en la lista
stop_client	-	Detiene todos los clientes incluidos en la lista
restart_client	-	Reinicia todos los clientes incluidos en la lista

4.4 Ciclo de vida

Durante todas las fases de investigación, diseño y desarrollo de la herramienta se ha seguido la metodología de desarrollo ágil Scrum, con *sprints* de dos semanas. De esta forma se proponían una serie de objetivos a cumplir para dicho *sprint* que combinaban diferentes partes del desarrollo cuando era posible, para agilizar el trabajo y tener partes probables que representaran funcionalidades completa en el menor tiempo posible, aunque la herramienta no estuviera desarrollada en su totalidad.

4.4.1 Pruebas

La técnica elegida para el desarrollo fue la conocida como TDD. Se escribían casos de uso y pruebas unitarias de la herramienta que representaran funcionalidades mínimas, después se desarrollaba la parte del código que debería cumplir con esa funcionalidad y por último se comprobaba si se pasaban todas las pruebas para comprobar si el código estaba correcto o era necesario arreglar algún fallo.

Como se ha mencionado en el párrafo anterior, se realizaron dos tipos de test: unitarios y de integración.

- **Test unitarios:** Se realizaban constantemente, cada vez que se completaba una característica, aunque su relevancia o su complejidad fueran mínimos, para asegurar que cumplían con su funcionalidad y que no afectaban a otras partes del código. Para la elaboración de estas pruebas se utilizó la librería unittest de Python.
- **Test de integración:** Estos test se realizaban una vez terminado el desarrollo de cualquiera de las partes. Gracias a ellos se puede comprobar la interacción entre los diferentes módulos y asegurar que todo funciona correctamente y responde como se espera. Para la realización de estos test se ha utilizado la herramienta lettuce para Python.

Finalmente se realizaron otra serie de pruebas de la herramienta completa en un entorno simulado. Se procedió, en primer lugar a desplegar trece máquinas en Amazon EC2, en las cuales, utilizando únicamente la herramienta desarrollada, se desplegó un clúster Etcd y diez clientes monitorizando los cambios. Después se procedió a escribir algunas claves y comprobar que todos los clientes se actualizaban correctamente. Estas pruebas, aunque no han sido frecuentes, sí que ha sido necesario realizarlas más de una vez para asegurar que todo funcionaba.

4.4.2 Gestión de la configuración

Para el control de versiones durante el desarrollo se utilizó la herramienta Git, subiendo todos los cambios realizados a un repositorio privado alojado en el sitio Github. El sistema Git se ha utilizado siguiendo las recomendaciones especificadas en el esquema de desarrollo conocido como GitFlow [14]. Para el tema de los *tickets* y las *issues* durante el desarrollo se ha utilizado el propio sistema GitHub, que ofrece estas capacidades como extensión al uso de Git para facilitar el trabajo de los desarrolladores.

5. Conclusiones y líneas futuras

A día de hoy las tecnologías de computación en la nube y los sistemas y aplicaciones distribuidos están experimentando un enorme crecimiento y están llamando la atención de multitud de empresas de todo el mundo. Es importante tener que en cuenta que para poder cumplir con las cada vez más exigentes demandas de los clientes es de vital importancia mantener el sistema en un estado óptimo y trabajando a pleno rendimiento en todo momento.

Para poder cumplir con esos objetivos no basta con desplegar las aplicaciones y ofrecer el servicio, es necesario mantener las máquinas trabajando y hacerlo de la manera más eficiente posible. Por ello, aunque no parezcan muy vistosas, las herramientas que proveen cualquier tipo de funcionalidad capaz de facilitar el trabajo a las personas encargadas de estos macro despliegues tienen un gran valor para las empresas, ya que suponen un ahorro en todos los sentidos.

Este proyecto se ha desarrollado teniendo en cuenta desde el primer momento a las personas que van a estar a cargo del funcionamiento de estas aplicaciones distribuidas. Se han propuesto varias soluciones posibles, pero siempre abogando por la utilización de las herramientas y sistemas más actuales y potentes en el campo del control de la información distribuida, para proporcionar la mejor experiencia posible al usuario en el futuro cercano.

La herramienta se encuentra en un punto de madurez suficiente para ofrecer un servicio básico a quien quiera utilizara, especialmente teniendo en cuenta que no existen muchas herramientas que cumplan con una finalidad similar. Sin embargo aún le queda un largo

camino por recorrer, pero tiene el potencial, siempre y cuando sea bien conducida para convertirse en una herramienta indispensable en el futuro.

A nivel personal este proyecto me ha aportado grandes conocimientos sobre el funcionamiento de los sistemas distribuidos y las principales preocupaciones que conlleva la administración de sistemas en la nube, así como permitirme descubrir herramientas muy útiles que no habría podido conocer de otra forma, como son las capacidades de los Amazon Web Services, o las bondades del lenguaje de programación Python. Este lenguaje ha supuesto un cambio radical en mi forma de pensar que se encontraba bastante limitada y encasillada por los escasos conocimientos de programación en los que había sido educado hasta ahora.

También agradezco la experiencia con herramientas que me han ayudado a desarrollar el proyecto, como el sistema de control de versiones Git, y la experiencia con administración de diversos tipos de máquinas Unix, que fueron necesarios para el despliegue de las máquinas en Amazon EC2 donde se realizaron todas las pruebas del sistema.

Finalmente, la posibilidad de trabajar en un entorno profesional real en una empresa de innovación tecnológica, como es Telefónica I+D, me ha proporcionado una gran visión de la forma de trabajo de las grandes empresas y de lo que me puedo encontrar en un futuro. He aprendido a trabajar en grupo y a conocer los procedimientos que es necesario seguir cuando se está trabajando en este tipo de entornos.

5.1 Líneas futuras

Durante el desarrollo surgieron una serie de cuestiones que se podrían mejorar del sistema, así como varias nuevas características que resultarían en una mejor experiencia para el usuario.

En primer lugar convendría mejorar el sistema de actualización de la configuración. En el estado actual solamente es posible escribir las claves una a una desde el manager, pero sería interesante en un futuro añadir la capacidad de subir archivos de configuración completos. Para esto será necesario implementar un conversor capaz de transformar el archivo de configuración en formato json al formato de claves y directorios propio de Etcd antes de escribirlas todas de una vez, y después incluir un

sistema similar en el cliente para realizar la operación inversa. Este mecanismo facilitaría en gran medida las tareas de actualización más grandes, así como el despliegue inicial de sistemas de configuración.

En segundo lugar parece interesante estudiar la posibilidad de ampliar la funcionalidad de los clientes. Como primera mejora se propone añadir la capacidad de escribir en el servidor de configuración, de esta manera se conseguirá una experiencia interactiva desde la aplicación distribuida, que permitirá a las diferentes máquinas comunicarse entre ellas. También se propone modificar la funcionalidad del cliente para que se dedique a monitorizar las claves concretas que necesite en lugar de descargar la configuración a un fichero cada vez que se produce un cambio. El principal problema de esta aproximación es que esto implicaría que la aplicación necesita conocer el cliente e interactuar con él constantemente, limitando en gran medida la libertad que se consigue con el estado actual de la herramienta.

Como tercer punto interesante cabe mencionar que sería muy interesante añadir capacidades de tenencia múltiple al sistema. El concepto de tenencia múltiple o Multitenancy define una forma de operación del software en el que múltiples instancias independientes de una o varias aplicaciones operan en un entorno compartido. Cada una de esas instancias se denomina *tenant*, y está separada a nivel lógico, pero comparte un mismo espacio físico, manteniendo la impresión para los diferentes *tenants* de que existe una instancia particular para cada uno de ellos [13]. Esta característica añadiría muchas posibilidades al sistema, permitiendo que un único despliegue de clúster Etcd atienda las necesidades de configuración de más de una aplicación. Por supuesto todo se podría seguir controlando de forma centralizada desde el manager, creando la herramienta perfecta para manejar la configuración en empresas que se encuentren en la situación de disponer de múltiples aplicaciones distribuidas que estén funcionando simultáneamente, permitiendo una importante reducción de costes así como aumentando el control sobre las aplicaciones. Además, gracias a la inclusión de esta característica la herramienta podrá ser ofrecida, ya no sólo como una herramienta de configuración, sino como un servicio de almacenamiento de configuración distribuida, similar a los servicios en la nube que ofrece Amazon Web Services.

La cuarta propuesta consiste en la implementación de un cliente web desde el que se controle por completo la herramienta. Como la herramienta dispone de unas funcionalidades bastante sencillas será relativamente fácil incorporarlas todas ellas en el cliente web, sin limitar las posibilidades que ofrece. La creación de este cliente web mejoraría en gran medida la experiencia de usuario, reduciendo la curva de aprendizaje, además de hacer la herramienta mucho más atractiva para el uso por todo tipo de usuarios, y no ya sólo desarrolladores y administradores de sistema, aunque estos siguen siendo el objetivo principal.

Como quinta propuesta se quiere indicar que resultaría de gran interés incluir la posibilidad de monitorizar de forma sencilla tanto el estado de la configuración en el clúster Etcd, como el estado de las propias máquinas que forman el clúster y también el estado de los clientes integrados en la aplicación distribuida. Con esta monitorización se podrá comprobar en todo momento que la configuración se encuentra en el estado deseado y que todas las máquinas están funcionando correctamente, para en caso contrario solucionar los problemas con la mayor brevedad posible. En el caso de que se llegara a crear el cliente web mencionado en el párrafo anterior se podría incluir un monitor en tiempo real dentro del mismo, permitiendo observar el funcionamiento de las máquinas de forma cómoda.

La última propuesta consiste en realizar una integración completa con la otra herramienta desarrollada en un TFG de forma paralela, capaz de realizar despliegues de aplicaciones distribuidas. De momento las dos herramientas presentan una funcionalidad independiente mientras conservan la capacidad de interactuar entre ellas, sin embargo este proceso requiere alguno pasos manuales aún que sería muy interesante eliminar de cara a una siguiente etapa de desarrollo.

Seguramente, si se continúa con el desarrollo de la herramienta surjan muchas más propuestas de mejora, pero de momento parece que las mencionadas en este capítulo representan el mejor camino a seguir por ahora.

Bibliografía

[1] A.B. Downey, *Think Python: How to Think Like a Computer Scientist*. O'Reilly, 2012.

[2] *Fabric*, http://www.fabfile.org/, 2014. Obtenido por última vez el 6/6/2014.

[3] *Amazon Web Services*, http://aws.amazon.com/es/, 2014. Obtenido por última vez el 6/6/2014.

[4] G.F. Coulouris, J. Dollimore & T. Kindberg, *Distributed systems: concepts and design*. Harlow, England, Pearson Education, 2005.

[5] L. Navarro, *Sincronización, tolerancia a fallos y replicación*. Universidad Oberta de Catalunya UOC, 2009.

[6] *Algoritmo de Paxos*, http://es.wikipedia.org/wiki/Algoritmo_de_Paxos, 2014. Obtenido por última vez el 6/6/2014.

[7] D. Ongaro & J. Ousterhout, *In Search of an Understandable Consensus Algorithm (Extended Version)*. Stanford University, 2013.

[8] B. Johnson, *Raft – The Understandable Distributed Protocol* http://www.infoq.com/presentations/raft, 2013. Obtenido por última vez el 6/6/2014.

[9] *Zookeeper Overview*, http://zookeeper.apache.org/doc/trunk/zookeeperOver.html, 2013. Obtenido por última vez el 6/6/2014.

[10] *Doozerd*, https://github.com/ha/doozerd, 2013. Obtenido por última vez el 6/6/2014.

[11] *Etcd*, https://github.com/coreos/etcd, 2014. Obtenido por última vez el 6/6/2014.

[12] *Long Polling*, http://en.wikipedia.org/wiki/Push_technology#Long_polling, 2014. Obtenido por última vez el 6/6/2014.

[13] Gartner. *Gartner IT Glossary: Multienancy*, http://www.gartner.com/it-glossary/multitenancy, 2013. Obtenido por última vez el 6/6/2014.

[14] *A succesful Git branching model, http://nvie.com/posts/a-successful-git-branching-model/, 2010.* Obtenido por última vez el 6/6/2014.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 20:47:28 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)